PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-057195
(43)Date of publication of application : 25.02.2000

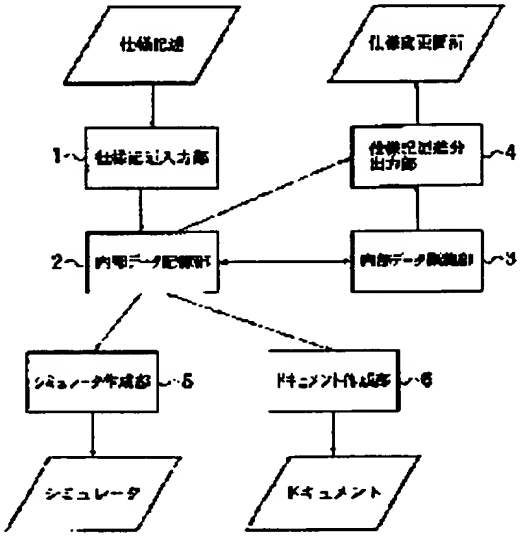
(51)Int.CI. G06F 17/50
G06F 9/38
G06F 11/28

(21)Application number : 10-228268
(22)Date of filing : 12.08.1998

(71)Applicant : TOSHIBA CORP
(72)Inventor : MIZUNO ATSUSHI

(54) PROCESSOR DESIGN SUPPORTING DEVICE

(57)Abstract:
PROBLEM TO BE SOLVED: To make attainable correct design in accordance with specifications and shortening of the design period by unifying the design description.
SOLUTION: The processor design supporting device is provided with a specification description input part 1 for converting the specification description of a circuit into internal data, an internal data storage part 2 for storing the internal data, a simulator preparing part 5 for extracting information required for simulation from the internal data to prepare the simulation and a document preparing part 6 for extracting information required for a document from the internal data to prepare the document. By the configuration, the both of the simulator and the document are prepared from one specification discription.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－57195
（P2000－57195A）
(43)公開日　平成12年２月25日(2000.2.25)

(51)Int.Cl.⁷		識別記号	FI		テーマコード（参考）
Ｇ０６Ｆ	17/50		Ｇ０６Ｆ	15/60	６５４Ａ　５Ｂ０１３
	9/38	３１０		9/38	３１０Ｘ　５Ｂ０４２
	11/28	３４０		11/28	３４０Ｃ　５Ｂ０４６

審査請求　未請求　請求項の数１　ＯＬ　（全 11 頁）

(21)出願番号　特願平10－228268

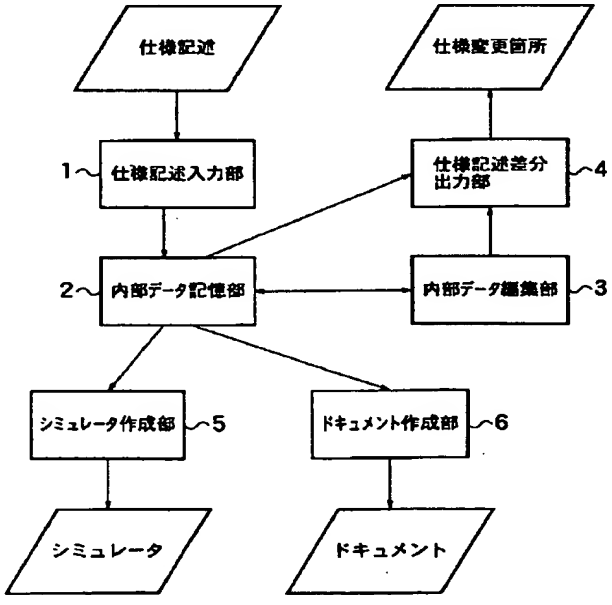
(22)出願日　平成10年８月12日(1998.8.12)

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　水野　淳
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内
(74)代理人　100083806
弁理士　三好　秀和　（外３名）

Ｆターム(参考)　5B013 AA20
5B042 HH38
5B046 AA08 BA02 JA01 KA10

(54)【発明の名称】　プロセッサ設計支援装置

(57)【要約】
【課題】　本発明は、設計記述を統一することにより仕様に応じた正確な設計ならびに設計期間の短縮化を達成することを課題とする。
【解決手段】　本発明は、回路の仕様記述を内部データに変換する仕様記述入力部１と、内部データを記憶する内部データ記憶部２、内部データからシミュレーションに必要な情報を抽出してシミュレータを作成するシミュレータ作成部５と、内部データからドキュメントに必要な情報を抽出してドキュメントを作成するドキュメント作成部６を備え、一つの仕様記述からシミュレータとドキュメントの双方を作成して構成される。



【特許請求の範囲】
【請求項１】　設計対象回路の命令セットアーキテクチャ、命令定義、データパス定義、パイプライン定義からなる仕様記述を入力し、入力された仕様記述を命令定義内部データ、パイプライン制御テーブルからなる内部データに変換する仕様記述入力部と、
前記仕様記述入力部で作成された内部データを記憶する内部データ記憶部と、
前記仕様記述入力部で作成されて前記内部データ記憶部に記憶された内部データを変更して編集する内部データ編集部と、
前記内部データ編集部により変更された変更後の仕様記述と変更前の仕様記述との差を仕様変更箇所として出力する仕様記述差分出力部と、
前記仕様記述入力部で作成されて前記内部データ記憶部に記憶された内部データからシミュレーションに必要な情報を抽出してシミュレータを作成するシミュレータ作成部と、
前記仕様記述入力部で作成されて前記内部データ記憶部に記憶された内部データからドキュメントに必要な情報を抽出してドキュメントを作成するドキュメント作成部とを備え、
一つの仕様記述からシミュレータとドキュメントの双方を作成することを特徴とするプロセッサ設計支援装置。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、統一された仕様記述によりプロセッサの設計を支援するプロセッサ設計支援装置に関する。
【０００２】
【従来の技術】従来のマイクロプロセッサの設計では、命令セットアーキテクチャの仕様を決定し、次にマイクロアーキテクチャやパイプライン構成などの実装の仕様を定義して、ＲＴＬ（レジスタトランスファレベル）記述を作成する。設計期間の短縮化や早期のモデル提供の要求から、ＲＴＬ設計を行う前の段階で性能見積もりや検証が重要となっており、命令セットアーキテクチャの性能を見積もるための命令シミュレータや、パイプラインレベルでの性能を見積もるためのサイクルアキュレートシミュレータが作成される。これらのシミュレータは異なる設計者が作成することが多く、シミュレーションプログラムのソースコードは作成したものでなければ把握できない場合が多い。また、ＲＴＬ記述の作成もこれらとは別の設計者が作成することが多い。このため、仕様の共有化という意味で、設計者間の意志の疎通を図るため、自然言語や図を用いて記述されたドキュメントが必要になっていた。
【０００３】このように、一つのプロセッサの仕様に対してドキュメント、シミュレータのソースプログラムなどの複数の記述が存在する。このため、設計途中で仕様に変更が生じた場合には、すべての記述を変更しなければならず、非常に効率が悪かった。また、記述間に不一致が生じる可能性が高く、設計時における設計者間の情報交換の混乱などにより設計期間のロスが生じることも多かった。さらに、仕様と実装設計の不一致、仕様とモデルの不一致なども発生していた。
【０００４】これに対処する手段としては、例えばプログラムの仕様を表すための基準となる記述が必要となる。しかし現状ではそれにふさわしい記述は存在しない。例えばドキュメントは自然言語で書かれているため曖昧さが生じ、計算機処理に不向きなため、記述のチェックは人手でしなければならない。一方、ＲＴＬ記述ではパイプラインの動作などを把握することは難しい。シミュレーション用の記述を含む動作レベルの記述は、現状ではプログラムの各機能を表現する決まった書き方は存在せず、情報交換や再利用には向いていない。このため、上記の不具合は解決できず、設計の効率化は実現できなかった。
【０００５】
【発明が解決しようとする課題】以上説明したように、従来における大規模なプロセッサの設計では、ドキュメント、ＲＴＬ記述、シミュレーション用記述などの複数の記述が存在するため、設計途中で仕様に変更が生じた場合には、記述変更作業にかかる手間や、記述間の不一致による設計者間の情報交換時の混乱など、設計期間の大幅なロスが発生し、また仕様と実装設計の不一致、仕様とモデルの不一致などの不具合を招いていた。
【０００６】そこで、本発明は、上記に鑑みてなされたものであり、その目的とするところは、設計記述を統一することにより仕様に応じた正確な設計ならびに設計期間の短縮化を達成し得るプロセッサの設計支援装置を提供することにある。
【０００７】
【課題を解決するための手段】上記目的を達成するために、請求項１記載の発明は、設計対象回路の命令セットアーキテクチャ、命令定義、データパス定義、パイプライン定義からなる仕様記述を入力し、入力された仕様記述を命令定義内部データ、パイプライン制御テーブルからなる内部データに変換する仕様記述入力部と、前記仕様記述入力部で作成された内部データを記憶する内部データ記憶部と、前記仕様記述入力部で作成されて前記内部データ記憶部に記憶された内部データを変更して編集する内部データ編集部と、前記内部データ編集部により変更された変更後の仕様記述と変更前の仕様記述との差を仕様変更箇所として出力する仕様記述差分出力部と、前記仕様記述入力部で作成されて前記内部データ記憶部に記憶された内部データからシミュレーションに必要な情報を抽出してシミュレータを作成するシミュレータ作成部と、前記仕様記述入力部で作成されて前記内部データ記憶部に記憶された内部データからドキュメントに必

要な情報を抽出してドキュメントを作成するドキュメント作成部とを備え、一つの仕様記述からシミュレータとドキュメントの双方を作成することを特徴とする。

【０００８】

【発明の実施の形態】以下、図面を用いてこの発明の一実施形態を説明する。

【０００９】図１はこの発明の一実施形態に係るプロセッサ設計支援装置の構成を示す図である。

【００１０】図１において、この実施形態のプロセッサ設計支援装置は、入力された仕様記述を内部データに変換し、入力となる仕様記述にパイプラインの定義がある場合は、各命令の動作をパイプラインステージごとの動作に分割し、依存関係によるハザードが発生する可能性のある命令の組に対して、停止する条件、停止させるタイミングなどを格納したパイプライン制御テーブルを作成する仕様記述入力部１と、仕様記述入力部１で生成された内部データを記憶する内部データ記憶部２と、内部データ記憶部２に記憶された内部データを表示し設計者が修正する機能を有し、パイプライン制御テーブルを直接変更してプロセッサの仕様を変更できる内部データ編集部３と、修正された内部データと入力の仕様記述との差分を出力し、内部データ編集部３で変更された部分と元の仕様の差分を出力することにより元の仕様と変更後の内部データの整合を持たせる仕様記述差分出力部４と、内部データ記憶部２に記憶された内部データからシミュレーションに必要な情報を抽出し、命令シミュレータ、サイクルアキュレートシミュレータのいずれも作成できるシミュレータを作成するシミュレータ作成部５と、内部データ記憶部２に記憶された内部データからドキュメントに必要な情報を抽出し、特にパイプライン制御テーブルに登録された命令の組のパイプライン上の動作を表したパイプライン図を作成し、命令セットアーキテクチャ、命令動作などを含めたドキュメントを作成するドキュメント作成部６を備えて構成され、プロセッサの仕様記述を入力とし、シミュレータとドキュメントを作成するシステムであり、一つの記述入力からシミュレータとドキュメントの両方を作成することを特徴とする。ここでドキュメントとは、命令セットアーキテクチャ、各命令の動作、パイプラインの構成、入力されたコード列に対するパイプライン動作の代表的な例などが書かれており、プロセッサ設計者やそのプロセッサ上で動作するアプリケーションソフトウェア開発者が、プロセッサの動作を読んで理解することを目的として作成された文書のことである。

【００１１】設計対象となるプロセッサの仕様記述は、命令セットアーキテクチャ定義、命令定義、データパス定義、パイプライン定義の各項目からなるとする。命令セットアーキテクチャ定義では、命令の形式、レジスタセットなどを定義する。命令定義は各命令の動作が定義されており、記述例は例えば図２に示すようになる。データパス定義は、プロセッサのデータパスを構成するハードウェアリソースを定義する。ここでハードウェアリソースは、すでに設計された具体的な回路でもよいし、詳細設計は未定であるが所望の機能を実行する抽象的な回路ブロックでもよい。ハードウェアリソースには機能とその内容（動作）が定義されているものとする。また、各動作にかかるサイクル数なども記述され、記述例は例えば図３に示すようになる。パイプライン定義は、ステージの定義とパイプライン制御の定義からなる。ステージの定義は各ステージの動作を定義したものであり、図４に示すように、ハードウェアリソースの動作を指定することにより定義されている。パイプライン制御の定義は、パイプライン上の命令の動きを定義したものであり、ステージの順序構成、ハザード発生時の動作などが定義されており、例えば図５に示すように定義される。

【００１２】仕様記述入力部１は、これらの記述を入力し、内部データへと変換する。内部データは大きく分けて、命令定義内部データとパイプライン制御テーブルからなる。命令定義内部データの一例を図６に示す。仕様記述入力部１は、図７に示すようなマイクロ動作対応表を保持しており、各命令の動作記述から一般のコンパイラ技術を用いて解析木に変換した後、解析木の各ノードに対応するマイクロ動作をマイクロ動作対応表から見つけだし、それに対応するハードウェアリソース機能とステージを抽出し、それらを一組にして一時的なリストに格納する。すべてのノードに対して処理を行った後、一時リストをステージ順にソートしてマイクロ動作単位の命令を順に命令定義内部データへと書き込む。

【００１３】例えば図２に示すＡＤＤ命令では、解析木のノードには、＝、×、［］が現れる。＝は図７に示すマイクロ動作対応表にないため何もしない。＋はＡＬＵのＡｄｄに対応するので図３より対応するコードを抽出し、さらにこれは図４よりＥＸステージに対応することがわかるので、それらをまとめて一時リストに格納する。［］は左辺に現れたか右辺に現れたかにより対応するマイクロ動作が異なるため解析時に右辺か左辺かの補助情報を付けておく。図７より右辺の［］はＧＰＲのＲｅａｄに対応する。同様な処理を行い一時リストに格納する。ＡＤＤ命令の全てのノードに対して処理を終えた後、一時リストをステージの順にソートし、図６に示すような動作コードを内部データに書き出す。図８に仕様記述の命令定義から命令定義内部データへと変換する処理フローを示す。

【００１４】内部データのもう一つの大きなデータはパイプライン制御テーブルである。パイプライン制御テーブルは、依存関係があったときの命令間の動作を定義したものであり、先行命令、後続命令、後続命令の停止ステージ、後続命令の停止条件、フォワーディング条件とフォワーディング動作の項目からなる。パイプライン制

御テーブルの一例を図９に示し、パイプライン制御テーブルの作成フローを図１０に示す。

【００１５】例えば、先行命令、後続命令ともＡＤＤ命令であった場合について説明する。ＡＤＤ命令はレジスタ読み出し動作もレジスタ書き込み動作もあるので、二つのＡＤＤ命令間にはデータ依存関係が存在する可能性がある。パイプラインのステージ単位の動作に分解された命令定義内部データから、レジスタ読み出し動作はＩＤステージで実行されることが求まり、後続命令の停止ステージにはＩＤステージをセットする。同様にしてレジスタ書き込み動作がＷＢステージで実行されることも求まり、フォワーディング機能がなければ先行命令のＷＢステージ未了が後続命令の停止条件となる。しかし、設計対象となるプロセッサにはフォワーディングの機能を持たせることがパイプライン制御部で定義されているため、フォワーディングを考慮した処理を行う。先行命令の結果のデータが確定するのはＡＬＵ演算が終了するＥＸステージである。フォワーディング機能があるため、先行命令のＡＬＵ演算の結果は次のサイクルで利用可能であり、後続ＡＤＤ命令はＩＤステージで停止する必要はない。すなわち、先行ＡＤＤ命令がＥＸステージを実行するのと同サイクルで、後続ＡＤＤ命令はＩＤステージを実行してよい。したがって、後続命令の停止条件は「先行命令がＥＸを未了」となる。後続命令がＩＤを実行するとき、先行ＡＤＤ命令のレジスタ書き込み動作が終了していなければ、フォワーディングしなければならない。したがって、フォワーディング実行条件は「先行命令がＷＢが未了」となり、フォワーディング動作は先行命令のＡＬＵ演算結果を後続命令のレジスタ読み出しデータとする。

【００１６】パイプライン制御テーブルは内部データ編集部３により設計者に対し表示される。設計者は図９に示すパイプライン制御テーブルをみて、特殊な制御が必要な命令が見つかるかもしれない。このような場合は、内部データ編集部３により内部データを直接書き換える。例えば図９において、ＬＯＡＤ命令はロードデータをフォワーディングしないという仕様に変更したいとする。その場合は、先行命令がＬＯＡＤの行の後続命令の停止条件を「先行命令がＷＢを未了」に、フォワーディング実行条件とフォワーディング動作の項目をそれぞれ「なし」に変更し、パイプライン制御テーブルは図１１に示すように変更される。仕様記述差分出力部４はこのような変更前後の項目の内容とを、例えば図１２に示すように組にして仕様変更箇所として出力する。

【００１７】シミュレータ作成部５は、データベースの中からシミュレーションに必要な部分を抽出し、結合してシミュレータを作成する。命令シミュレータを作成する場合とサイクルアキュレートシミュレータを作成する場合ではフローが異なる。命令シミュレータを作成する場合は、内部データからレジスタメモリに対応する変数を決定し、フェッチ、デコード、命令実行を繰り返すコードを生成する。命令実行は、命令の種類を条件節として対応する命令の動作を実行するコードを作成する。命令定義内部データの各命令の動作から、パイプラインステージを無視して生成する。命令シミュレータのコードは例えば図１３に示すようになる。

【００１８】一方、サイクルアキュレートシミュレータを作成する場合には、内部データから、レジスタメモリに対応する変数を決定し、１サイクルの実行を繰り返すメインルーチンのコードを作成する。その後フェッチとデコードのステージのコードを作成する。命令デコードより後のステージは一旦リストに格納され、そのリストを順に各ステージのコードを作成する。ステージのコードは、まず実行すべき命令に対して依存関係のある命令を取り出す関数を呼び出すコードを書き、その後各命令のコードを条件文の各節に記述する。各命令に対しては、パイプライン制御テーブルから自分が後続命令であるときの停止条件やフォワーディング実行条件により実行するコードを作成して書き出し、それらが成立しない場合には通常の命令動作を行うコードを書き出す。例えば図１４に示すようなサイクルアキュレートシミュレータのコード例では、ＩＤステージに対する実行関数のヘッダ部をまず書き出し、その次に依存関係のある命令を抽出する関数呼び出しを書き出す。次に、命令定義内部データの命令を順に取り出す。ＡＤＤの場合には、実行命令がＡＤＤである場合のｃａｓｅ文のタグを書き出し、依存関係のある命令が存在しないという条件節を書き出し、ＡＤＤ命令のステージＩＤの実行コードを書き出す。次に、依存関係のある命令が存在したときの処理のコード生成部の書き出しに移る。パイプライン制御テーブルを順にチェックし、後続命令としてＡＤＤが一つでもあれば、依存関係のある先行命令Ｐに対する条件分岐コードを作成する。図９に示すパイプライン制御テーブルには、後続命令がＡＤＤ、先行命令がＡＤＤの行が存在するので、それに対するコードを次に書き出す。先行命令がＡＤＤであるときの条件タグを書き、次にフォワーディング実行条件があるので、その条件を条件節として書き出す。ここでは、先行命令ｐがステージｓを未了という判定を行う関数not_finishiedが用意されているとする。そして実行部分にフォワーディング動作を書き出す。以下フォワーディング条件が不成立で停止条件が成立している場合のコードと続き、これを後続命令がＡＤＤであるすべての行に対して繰り返す。さらに、すべての命令、すべてのステージに対して同様の処理を繰返し、コードを書き出す。ＥＸステージのように、パイプライン制御テーブルに無関係のステージに対しては、依存関係のある先行命令を抽出するコードを削除してから書き出してもよい。また、バイナリコードとして書き出してもよい。図１５にサイクルアキュレートシミュレータの作成フローを示す。

【００１９】ドキュメント作成部６は、命令セットアーキテクチャ、各命令の動作、パイプラインの動作を記述したドキュメントを内部データから作成する。命令の動作のドキュメントの記述例を図１６に示す。パイプライン動作のドキュメントでは、ステージの構成の他に、図１７に示すような命令の組み合わせごとのパイプライン動作をパイプライン図として表す。パイプライン図の作成フローを図１８に示す。パイプライン制御テーブルのすべての命令の組に対して、パイプライン図を作成する。

【００２０】図９に示すＡＤＤ命令同士の組み合わせを例に説明する。先行命令のＡＤＤが実行するステージを順に、ＩＦ、ＩＤ、ＥＸ、ＤＭ、ＷＢとファイルに書き出す。続けて１サイクル後に後続命令のＡＤＤがパイプラインに投入されたとして、先行命令のパイプライン図より１ステージずらして、先行命令のＩＤの真下に後続命令のＩＦがくるようにする。１ステージを書き出すごとにパイプライン制御テーブルを参照し、後続命令の停止条件をチェックする。図９に示すパイプライン制御テーブルでは、後続命令の停止ステージはＩＤであるので、ＩＦはそのまま書き出す。次のＩＤは後続命令の停止ステージであるので、フォワーディング実行条件と後続命令の停止条件を参照する。フォワーディング実行条件を満たし、後続命令の停止条件は満たしていないため、後続のＡＤＤはフォワーディングによりそのまま実行が続けられる。このため、後続命令は停止せず、先行命令と１サイクル分ずれたパイプライン図をそのまま書き出す。次に、先行命令がＬＯＡＤ命令で、後続命令がＡＤＤ命令の場合には、内部データ編集部３で図１１に示すように変更されているのでフォワーディング実行条件が存在しない。このため後続命令の停止条件だけを参照することになり、先行命令がＷＢを終了しない間は、後続のＡＤＤ命令はＩＤステージのまま停止記号（本実施例では－）を書き出し続けることになる。先行命令のＷＢの位置より右にきた時点で、ＩＤステージ以下を書き出す。この処理を、パイプライン制御テーブルの全ての命令の組みに対して繰り返す。このように、統一された記述に基づいて、入力された仕様記述から内部データが作成され、内部データからシミュレータならびにドキュメントが作成され、また仕様が変更された場合の変更前後の内容が出力されてプロセッサの設計が行われていく。

【００２１】したがって、上記実施形態によれば、一つの記述からシミュレータとドキュメントの双方が作成されるため、仕様、シミュレーションモデル、ドキュメントの不一致は回避され、設計の効率向上につながる。また、各命令の動作をパイプラインの実装に依存しない記述から、内部でパイプラインステージ単位の動作に分離してサイクルアキュレートなモデルに変換するため、パイプラインの仕様が変更になっても各命令の動作を変更する必要がなく、設計の効率向上につながる。命令シミュレータとサイクルアキュレートシミュレータを同時に作成することもできる。

【００２２】さらに、内部データ編集部３によりパイプラインの制御の詳細な部分を修正、変更することができるようになり、初期の仕様記述ではできなかった例外的な動作や、解析してから初めて気づくような動作の処理を付加することができる。また、仕様記述差分出力部４により内部データを変更した部分とそれに対応した元の仕様を出力することにより、オリジナルの仕様と内部データ変更後のデータの不一致を回避することができる。

【００２３】

【発明の効果】以上説明したように、この発明によれば、統一された記述によりプロセッサの設計を進めるようにしたので、仕様の変更に対して柔軟に対応することができ、仕様と実装設計、仕様とモデルとの不一致が回避できる。これにより、仕様に応じた正確な設計ならびに設計期間の短縮化を達成することができるようになる。

【図面の簡単な説明】

【図１】この発明の一実施形態に係るプロセッサ設計支援装置の構成を示す図である。

【図２】命令定義の記述例を示す図である。

【図３】ハードウェアリソースの一定義を示す図である。

【図４】パイプラインステージの一定義を示す図である。

【図５】パイプライン制御の一定義を示す図である。

【図６】命令定義内部データの一例を示す図である。

【図７】マイクロ動作対応表を示す図である。

【図８】命令定義内部データの作成フローを示す図である。

【図９】パイプライン制御テーブルの一例を示す図である。

【図１０】パイプライン制御テーブルの作成フローを示す図である。

【図１１】内容変更後のパイプライン制御テーブルを示す図である。

【図１２】仕様記述差分出力の一例を示す図である。

【図１３】命令シミュレータのコード例を示す図である。

【図１４】サイクルアキュレートシミュレータのコード例を示す図である。

【図１５】サイクルアキュレートシミュレータの作成フローを示す図である。

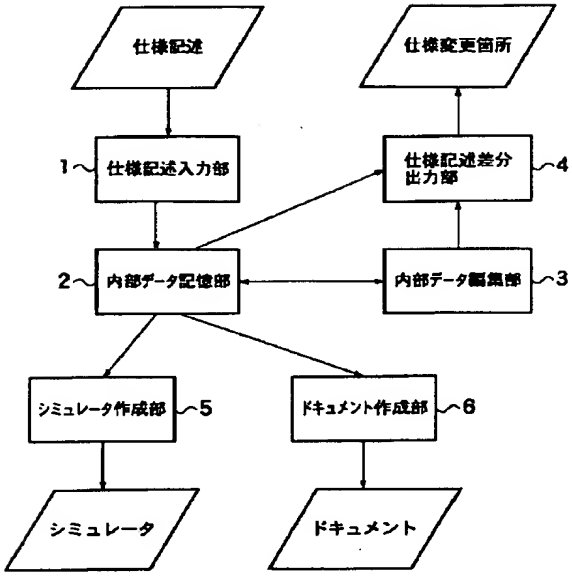
【図１６】命令のドキュメントの記述例を示す図である。

【図１７】パイプライン図の一例を示す図である。

【図１８】パイプライン図の作成フローを示す図である。

【符号の説明】
1　仕様記述入力部
2　内部データ記憶部
3　内部データ編集部
4　仕様記述差分出力部
5　シミュレータ作成部
6　ドキュメント作成部

【図１】



【図２】

```
ADD(rd,rs,rt)
      {    GPR[rd]=GPR[rs]+GPR[rt];   }
SUB(rd,rs,rt)
      {    GPR[rd]=GPR[rs]-GPR[rt];   }
LOAD(rt,offset,base)
      {    Vaddr=offset+GPR[base];
           Paddr=AddrTrans(Vaddr);
           GPR[rt]=Mem[Paddr];
      }
```

【図３】

リソース名	機能	入力	出力	動作	サイクル数
ALU	Add	A,B	C	C=A+B	1
	Sub	A,B	C	C=A−B	1
GPR	Read	D,I		GPR[I]=D	1
	Write	I	D	D=GPR[I]	1
Mem	Load	Addr	D	D=Mem[Addr]	1(キャッシュ・ヒット時)
					3(キャッシュ・ミス時)
	Store	Addr,D		Mem[Addr]=D	1(キャッシュ・ヒット時)
					3(キャッシュ・ミス時)

【図４】

ステージ	動作
IF	PC : CountUp IMem : Fetch
ID	Decode : Decode GPR : Read
EX	ALUのすべての機能
DM	Memのすべての機能
WB	GPR : Write

【図５】

ステージ順序構成	IF→ID→EX→DM→WB

データハザード解消動作	#　Fi：先行命令、Si：後続命令
	if (Result_has_Generate(Fi))
	　forward(Si.src,Fi.result)
	else
	　wait(Si);

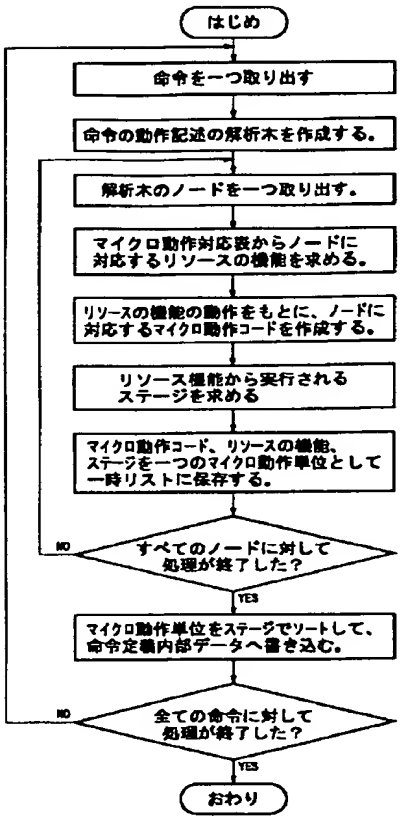
【図７】

ノード	リソース
[] (右辺)	GPR : Read
+	ALU : Add
[] (左辺)	GPR : Write

【図６】

命令名	命令形式	動作	リソース	ステージ
ADD	ADD rd rs rt	A=GPR[rs]； B=GPR[rt]； C=A+B； GPR[rd]=C；	GPR：Read GPR：Read ALU：Add GPR：Write	ID ID EX WB
SUB	SUB rd rs rt	A=GPR[rs]； B=GPR[rt]； C=A−B； GPR[rd]=C；	GPR：Read GPR：Read ALU：Sub GPR：Write	ID ID EX WB
LOAD	LOAD rt offset(base)	A=GPR[base]； Vaddr=offset+A； Paddr=AddrTrans(Vaddr)； D=Mem[Paddr]； GPR[rt]=D；	GPR：Read ALU：Add TLB：Trans Mem：Read GPR：Write	ID EX DM DM WB

【図８】



【図１６】

命令名	命令形式	動作
ADD	ADD rd rs rt	GPR[rd]=GPR[rs]+GPR[rt];
SUB	SUB rd rs rt	GPR[rd]=GPR[rs]−GPR[rt];
LOAD	LOAD rt offset(base)	Vaddr=offset+GPR[base]; Paddr=AddrTrans(Vaddr); GPR[rt]=Mem[Paddr];

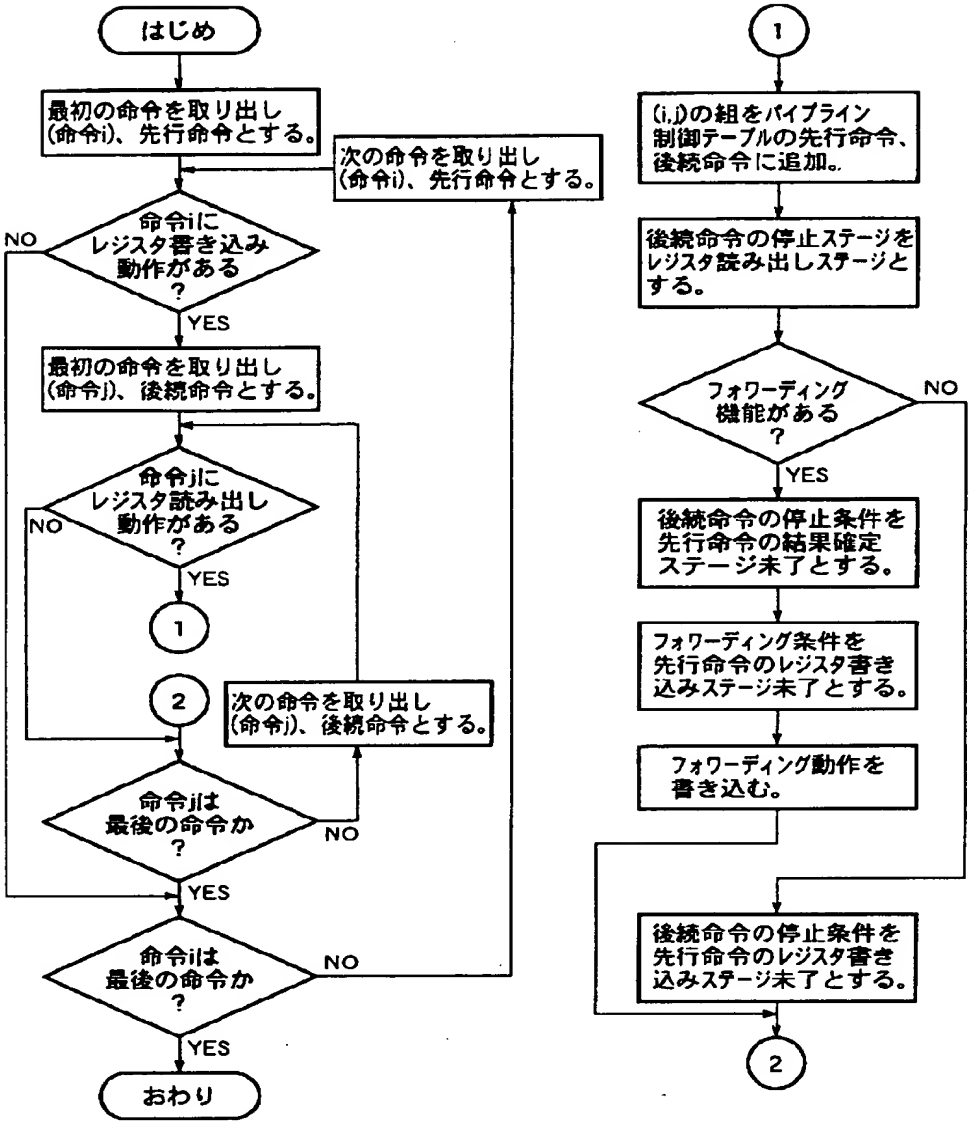
【図９】

先行命令	後続命令	後続命令の停止ステージ	後続命令の停止条件	フォワーディング実行条件	フォワーディング動作
ADD	ADD	ID	先行命令がEXを未了。	先行命令がWBを未了。	先行命令のALU演算結果を後続命令のレジスタ読み出しデータとしてバイパスする。
LOAD	ADD	ID	先行命令がDMを未了。	先行命令がWBを未了。	先行命令のロードしたデータを後続命令のレジスタ読み出しデータとしてバイパスする。

【図１１】

先行命令	後続命令	後続命令の停止ステージ	後続命令の停止条件	フォワーディング実行条件	フォワーディング動作
ADD	ADD	ID	先行命令がEXを未了。	先行命令がWBを未了。	先行命令のALU演算結果を後続命令のレジスタ読み出しデータとしてバイパスする。
LOAD	ADD	ID	先行命令がWBを未了。	なし。	なし。

【図１０】



【図１２】

	先行命令	後続命令	後続命令の停止ステージ	後続命令の停止条件	フォワーディング実行条件	フォワーディング動作
変更前	LOAD	ADD	ID	先行命令がDMを末了。	先行命令がWBを末了。	先行命令のロードしたデータを後続命令のレジスタ読み出しデータとしてバイパスする。
変更後	LOAD	ADD	ID	先行命令がWBを末了。	なし。	なし。

【図１３】

```
while ( ! end_of_program) {
  code=fetch(PC) ;
  I    =Decode(code) ;
  Execute(I,code) ;
}

Execute(I,code) {
  switch (I) {
    case ADD : A=GPR[code.rs] ;
               B=GPR[code.rt] ;
               C=A+B ;
               GPR[code.rd]=C ;
               break ;
    ...
}
```

【図１４】

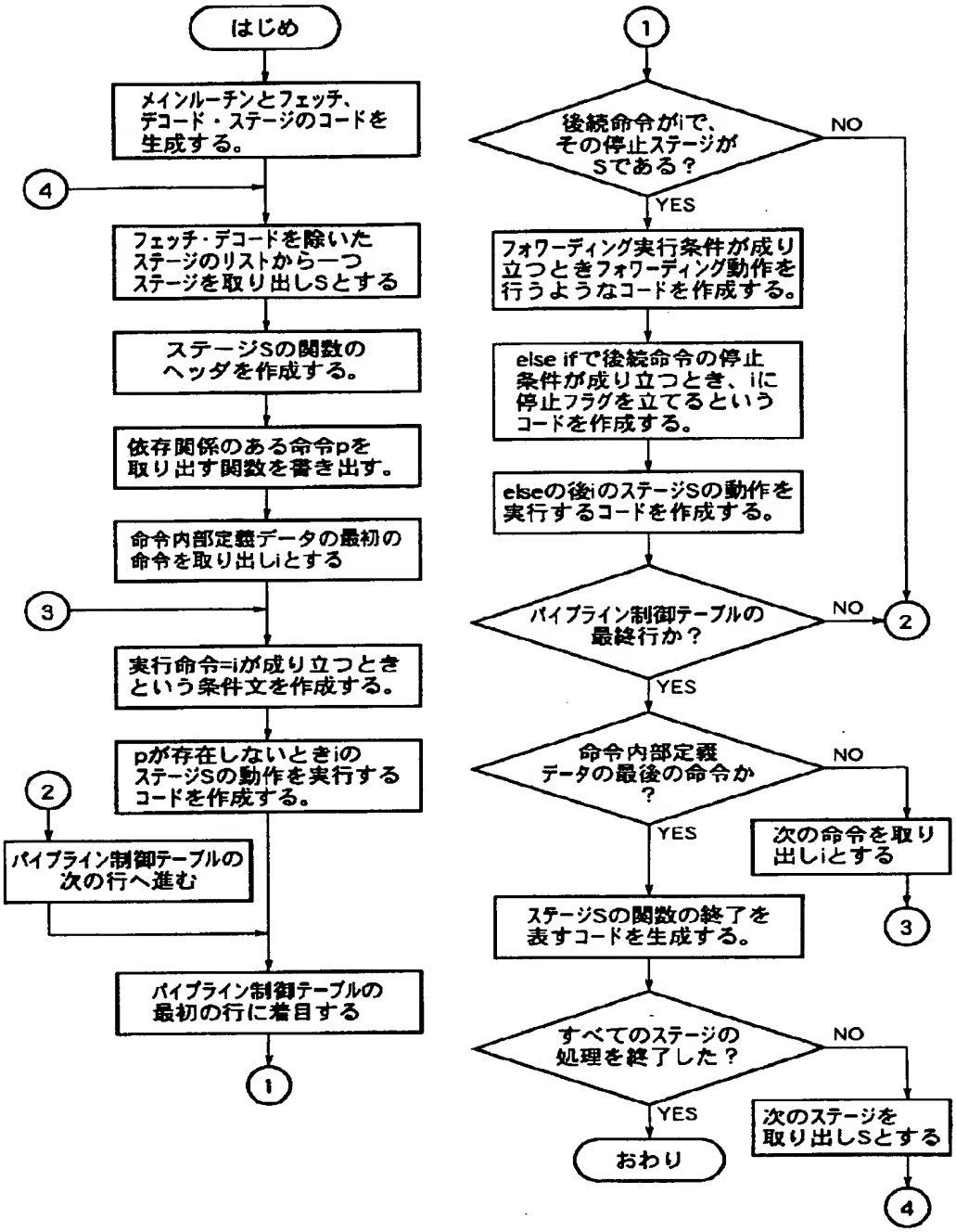
```
    while ( ! end_of_program) {
      IF_exec(code,PC) ;
      ID_exec(I,code) ;
      EX_exec(I,code) ;
      DM_exec(I,code) ;
      WB_exec(I,code) ;
    }
  - - - -
  - - - -
    ID_exec(I,code) {
      P=have_dependency(I,ID) ;
      switch (I) {
          case ADD :
            if (P==nil) {
              I.A=GPR[code.rs] ;
              I.B=GPR[code.rt] ;
              break ;
            }
            switch (P) {
              case ADD:
                if (not_finished(P,EX)) {
                  forward_ADD_ADD(I,P) ;
                }
                else if (not_finished(P,WB)) {
                  set_stall(I) ;
                }
                else {
                  I.A=GPR[code.rs] ;
                  I.B=GPR[code.rt] ;
                }
                break ;
              - - - -
      }
    }
    EX_exec(I,code) {
      switch (I) {
          case ADD : I.C=I.A+I.B ; break ;
          case SUB : I.C=I.A-I.B ; break ;
          ...
      }
```

【図１７】

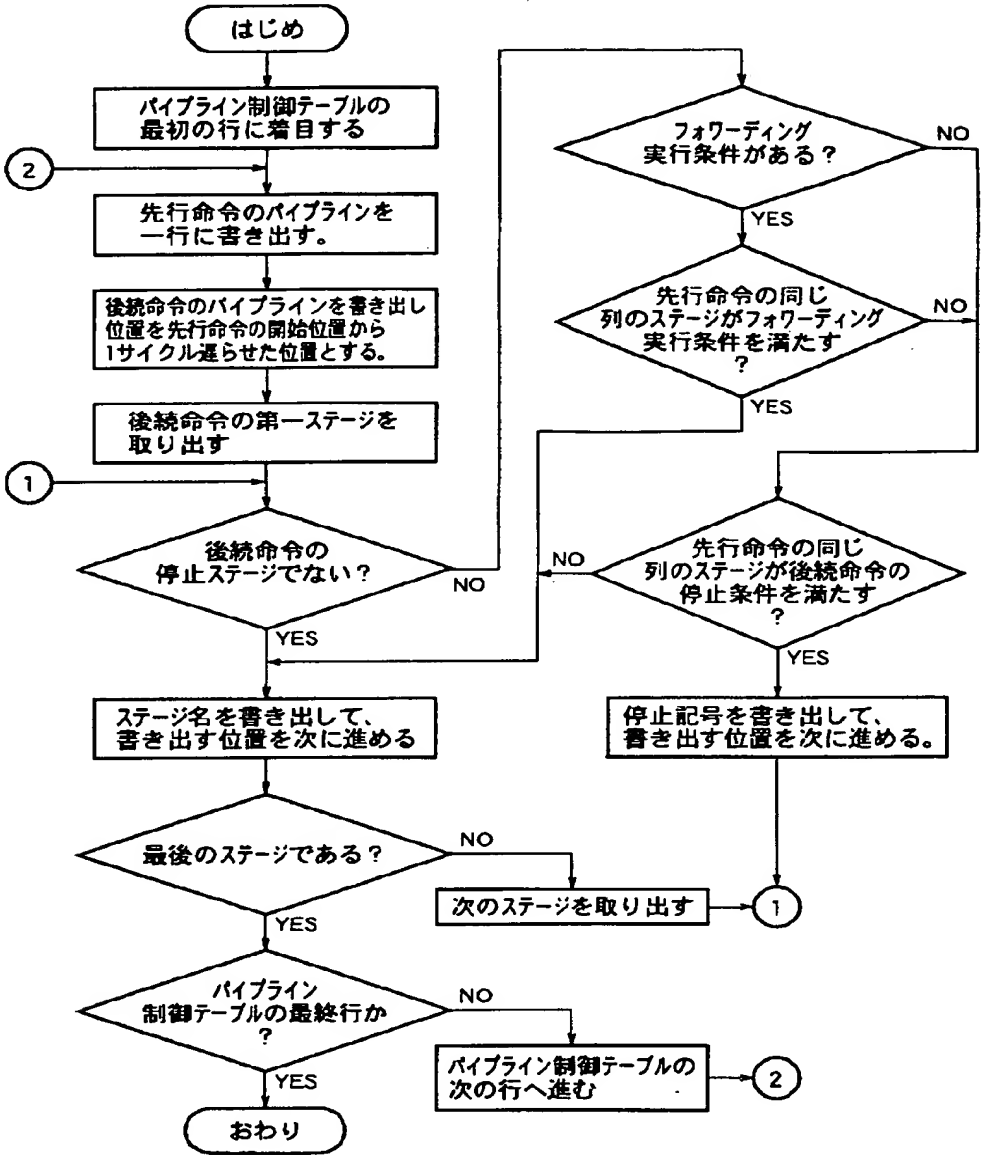
```
ADD   r2 r1 r1          | IF | ID | EX | DM | WB |
ADD   r4 r2 r3               | IF | ID | EX | DM | WB |

LOAD  r2 r1 0x000f      | IF | ID | EX | DM | WB |
ADD   r4 r2 r3               | IF | -- | -- | -- | ID | EX | DM | WB |
```

【図１５】


はじめ
メインルーチンとフェッチ、デコード・ステージのコードを生成する。
4
フェッチ・デコードを除いたステージのリストから一つステージを取り出しSとする
ステージSの関数のヘッダを作成する。
依存関係のある命令pを取り出す関数を書き出す。
命令内部定義データの最初の命令を取り出しiとする
3
実行命令=iが成り立つときという条件文を作成する。
pが存在しないときiのステージSの動作を実行するコードを作成する。
2
パイプライン制御テーブルの次の行へ進む
パイプライン制御テーブルの最初の行に着目する
1
1
後続命令がiで、その停止ステージがSである？
NO
YES
フォワーディング実行条件が成り立つときフォワーディング動作を行うようなコードを作成する。
else ifで後続命令の停止条件が成り立つとき、iに停止フラグを立てるというコードを作成する。
elseの後iのステージSの動作を実行するコードを作成する。
パイプライン制御テーブルの最終行か？
NO
2
YES
命令内部定義データの最後の命令か？
NO
YES
次の命令を取り出しiとする
3
ステージSの関数の終了を表すコードを生成する。
すべてのステージの処理を終了した？
NO
YES
次のステージを取り出しSとする
4
おわり

【図１８】


はじめ
パイプライン制御テーブルの最初の行に着目する
2
先行命令のパイプラインを一行に書き出す。
後続命令のパイプラインを書き出し位置を先行命令の開始位置から1サイクル遅らせた位置とする。
後続命令の第一ステージを取り出す
1
後続命令の停止ステージでない？
NO
YES
フォワーディング実行条件がある？
NO
YES
先行命令の同じ列のステージがフォワーディング実行条件を満たす？
NO
YES
先行命令の同じ列のステージが後続命令の停止条件を満たす？
NO
YES
ステージ名を書き出して、書き出す位置を次に進める
停止記号を書き出して、書き出す位置を次に進める。
最後のステージである？
NO
YES
次のステージを取り出す
1
パイプライン制御テーブルの最終行か？
NO
YES
パイプライン制御テーブルの次の行へ進む
2
おわり